*F1 - Intro*

* Kommentarer i källkod precis lika viktiga som koden.
* Identifierare används oftast för att ge namn åt klassen, variabler och metoder  
   \* Halva arbetet är att hitta bra namn på identifieraren
* Bryt ner uppgiften i mindre delar
* Klass -> definerar data och metoder
* Metoder -> Namngiven enhet med programkod (funktion, procecur)
* Objekt -> en instans av en klass (tar plats)
* Metodanrop, ->kör koden i en viss metod
* Signatur -> En metodsklass, namn och ordningsföljd på paramtertyperna

*F2 - Datatyper*

* Variabler är databehållare  
  varje variabel deklareas med en datatyp  
  int count = 0;
* Konstanter är variabler som inte kan ändras   
  Får sitt värde vid initaliseringen  
  Namnges med stora bokstver och underscore
* Primitiva datatyper innehåller tal  
  Datatypen bestämmer hur mycket som får plats och den tolkas vid läsning
* Referensdatatyp avser alltid en viss klass. Pekar på ett objekt. Om inte pekar det på null.
* <egenskaper> <datatyp> <identiderare> <initiering>  
  public int x  
  protected float count   
  private long myShape  
    
  % modulo (rest vid heltalsdivsion)

*F3 – Klasser och objekt*

* Klasser och deras instanser framställs ibland som simpla databehållare.  
  Klassen sammanför data med metoder.
* Instansvariabler skapas med objektet  
  De är tillgängliga så länge objektet är tillgängligt.  
    
  <åtkomst> <egenskap> <returdatatyp> <namn>(’parameterlista’) {…}  
  public none void <datatyp>  
  protected static <namn>  
  private abstract
* Metodens kropp är ett block. Inleds med { och avslutas med }  
  Innuti finns
* Lokala variabler - > existerar bara i blocket där de deklarerats
* Programsatser -> Aritmetiska uttryck/beräkning
* Block börjar alltid exekveras i main.
* **Static,** Konstanter, variabler och metoder kan placeras i static.  
  Finns bara en kopia i sin klass.  
  kan anropas utan att skapa objekt av klassen.
* Metoden Main måste vara static. Rena funktioner ska vara static & konstanter,  
  grundregel, undvik använda Static.

*F4 – Klasser och metoder*

* Klasser är huvuddelen i programmet(utgör idén)
* Objekt instantieras (utgör förekomsten)
* Programsater utförs enligt.  
  > Sekvens a; b; c; , väldefinerad ordning  
  > Selektion if(villkor) t-stmt; else f-stmt  
   >sätta upp ett villkor beroende på kommanda eller fil  
  > Iteration, while(villkor) a;, Medans uttrycket är uppfyllt
* Programsatser   
  > Tilldelning :> Variabel = uttryck  
  >Metodanrop  
  >Kontrollstruktur, (if, while,for,do)  
  > Satsblock {……}
* Iteration – for  
  for (<initiering>; <villkor> <uppdatering> {….}  
   Deklarera lokala true/false  
   variabler   
  Antalet varv är känt
* Iteration – while  
  while(<villkor> {….}  
  Villkoret prövas först, därefter utförs programsatsen så länge villkortet är sant.  
  Villkoret testas först.
* Iteration – do  
  do {….  
  } while (i < villkor2){…}  
  Kommer alltid göras en iteration för att sedan leta efter ett vilkor som uppfyller flera iterationer.-. Minst en gång genom loopen
* Oändliga loopar  
  Det är lätt hänt att man skriver loopar som pågår längre än man tänkt sig.  
  Ibland vill man ha en ”oändlig loop”  
  while (true) {..} -> Denna loop pågår tills programmet avslutas.
* Break; -> Hoppa till slutet av närmaste for, while, do eller switch
* Continue -> Hoppas till början av for, while eller do
* **Return;** Kompilatorna klagar om return fattas eller har fel typ
* Man kan ha flera retur-satser
* Om metodens typ är void, krävs inte return
* Varje metod har en signatur (kännetecken)
* Signaturen bestäms av metodens klass, namn och paramterern typ  
  Två olika metoder kan ha samma namn, klass, men olika parametrar.  
    
  Kompilatorn använder signaturen för att hitta rätt metod att anropa.
* Metodens returtyp ingår inte i signaturen.
* Konstruktorn anropas när instansen skapas.  
  Konstruktorn initierars och returner instansen och har därför samma typ som klassen.
* Man kan ha flera konstruktorer med olika parametrar.
* Public -> Synlig för alla klasser
* (Default) -> Synlig inom sitt paket
* protected -> synlig i underklasser
* Private -> Synlig i den egna klassen  
  ***(Börja restriktivt och öppna vid behov)***
* Klasser är nästan alltid public.
* Instansvariabler är oftast private eller protected.
* Metoder är oftast public eller protected.

*F5 – Selektion och Iteration*

* Boolska värden gör det möjligt att välja vilken kod som ska exekveras.  
  if ( x< 0){ if (n == 1)  
   x = -x: } s= things else(s = ”things”)
* == Likhetsoperatorn, Kräver exakt likhet, (alla bitar)
* Om två referenser jämförs så måste det vara samma referens för att det ska returnera true.
* Logiska operatorer tar boolska värden som argument
* !Negation -> !true get värdet false
* && Konjuktion -> Logisk OCH, båda sidor måste vara true
* || Disjunktion -> Logisk ELLER, en eller båda sidor måste vara true
* Används för kobinerade villkor.  
  if (done || (maxlimit < count)){..} //Alt 1  
  if (!done && (count > maxLimit) {…} //Alt 2
* **Satsblock,** Använd satsblock och identering för att undvika förvirring
* Med hjälp av satsblocken {..} kan flera programsatser behandlas som en enhet.
* Satsblock är ibland obligatoriska och rekommenderas i if-satser, for, while, och do-loopar.
* Villlkorsoperatorn. <villkor> ? t-värde : f-värde  
    
  Tecken kodas som 16-bitars positiva heltal  
  första 127 koderna = ASCII //Ameriskana tecken  
  första 256 koderna = ISO8859-1 //Nationell tecken
* Strängar är objekt; en referensdatatyp  
  Textsträngar är instanser av klassen String.  
  varje instans är slutgiltig, En ändrad String är en ny String.

*F6 – Objektorienterad Design*

* Skapa klasser, metoder, variabler > Viktiga delar i programmeringen.
* Kravspecifikation -> Vad ska programmet göra?
* Design -> Hur ska programmet göra det? (Kreativa fasen)
* Implementation -> Översätt design till programkod (Bygga efter ritning)
* Testning -> Verifiera att kraven är uppfyllda
* Design – Aggregering, Aggregerade objekt byggs upp med hjälpav andra objekt.
* **Interface-klasser(datatyp),** innehåller Konstanter och abstrakta metoder.Vilken i sin tur har signatur och returtyp men ingen kod.
* Ett interface specifierar ett beteende och används ofta.
* En klass som implementerar ett interface utlovar ett väldefinerat beteende.  
  För att använda objektet behöver man bara känna till interfacet  
  Interfacet löser problemet med att arv (extends) endast kan ärva från en klass.
* Interface kan ärva från andra interface men de, implemterade klassen måste implemterar all metoder som finns tillgängliga.
* Hjälpmetoder förenklar koden, Undviker repepition och redudans  
  Bryter ner i enklare delar: Men justeringar kan leda till nya fel. Automatiserad testing underlättar.   
  Black-box testning -> Skickar in känd indata och förväntar sig rätt utdata  
  White-box testning -> Tester är utformat för att gå alla väar genom koden.

*F7 – Arrayer*

* En array är en lista av vaiabler av samma typ. Arrayen har en fast längd (antal element) Alla element i en array har samma datatyp.
* Arrayer är Objekt, Instansen skapas med new int [8];  
  Array-variabeln är en arraytyp. Arrayen kan skapas från alla element.
* Indexering utanför arrayen får en ArrayIndexOfBoundsException.  
  Lägsta index = 0;Högsta = Array.lenght -1. *Arrayens längd finns i Array.length.*
* Om array-variabeln inte referar till ett array-objekt(null) så ges en NullPointerException.
* En array kan initeras vid skapandet och metoden kan ändras inom arrayen.   
  int temp = desk[1];  
  desk[2] = desk[2]
* Varje element är en referens till objekttypen, Varje element är null från början.  
  Referensen tilldelas, inte objektet.

*F8 – Arv*

* Klasser innehåller attribut och beteenden. En subklass ärver dessa från förälderna. Detta ger Återanvändbar kod och specialisering.
* Varje klass kan bara ärva från en förälder. Men en klass kan implementera flera interface.
* Man kan även Åsidosätta metoder (Overide methods). Subklassen åsidosätter föräldrars versioner.
* Super låter subklassen anropa föräldrarens version av metoden. Men detta gäller endast en nivå i klasshierarkin.
* Undvik skuggvariabler. En klass har variabler subklassen kan använda sig av dem och kan deklarera en ny variabel med samma namn/typ.
* Alla klasser ärver från Objekt.
* En abstrakt klass innehåller en eller flera abstrakta metoder. En abstrakt klass kan inte instantieras. Man man kan ärva från den.  
  Innehåller offullständig beskrivning av en klass.   
  Agerar som en mall för subklasser, eftersom när en subklass ärver från en abstract klass måste då alla metoder implemteras (overide) från den abstrakta metoden. Om inte blir subklassen också abstract.
* En subklass ska vara en mer specifik version av föräldern. Placera gemensama särdrag så högt upp i klasshierarkin som det är praktiskt.
* Skriva relevant kod med en balans och överdriv inte.
* Låt varje klass själva hantera sina interna data. Använd interface för att skapa en klass med flera beteenden /roller
* Designa klasshierakin för programmet men tänkt även framåt.  
  Använd åtkomstskydd (private/protected för att skapa inkapsling
* Använd final för att blockera arv och åsidosättning när det är motiverat.

*F9 – Polymorfism*

* Polymorfism innebär inom objektorienterad programmering att flera olika subklasser under en superklass kan ha ett gemensamt gränssnitt.  
    
  Det innebär att klasser med olika behov vad gäller implementering av en viss metod , ändå kan anropas på samma sätt. Den verkställande programkoden finns i respektive subklass, medan det gemensamma gränssnittet definieras i superklassen.
* Polymorfism – genom arv, Subklassen ärver och åsidosätter metoder  
  Med en referns till en generellare typ kan typens subklassen behandlas uniform.
* Polymorfism genom interface.  
  Subklasser har metoder med samma namn  
  En klass kan implemterar ett eller flera interface. Och enligt deklarationen måste alla metoder vara med.
* Polymorfism kan ge generell kod. Kod som kan användas till flera klassen underlättar återanvändningen. Onödiga detaljer kaplas in och ger en Renare design och tydligare struktur.
* En plymorfisk referens kan refera till objekt av olika typ under programmets exekvering.
* Bindningen mellan anrop och metod sker först vid exekvering.
* Vi styr vilken kod vi vill använda med polymorfism
* Polymorfism är en kraftfull mekanism och kräver god design.

*F10 - Exeption*

* En exeption är ett objekt. En exeption kastas uppåt längs anroppstacken när ett fel inträffar. Om en exception nät ut ut metoden main utan att fångas (catch) avslutas programmet.
* Vissa typer av exception kan förutses.  
  Try /Catch. >
* Det är inte alltid självklart på vilken nivå en exception hanteras bäst. Förutsägbara fel bör hanteras där de kan åtgärdas.

*F11 – Rekursion*

* Rekursion är en programmeringteknik.. En metod som anropar sig själv.  
  Kontrollerar om det finns ett enkelt svar (Stoppvillkor), Gör en del av arbetet och förminskar problemet samt anropar sig själv för att göra resten av problemet.
* Rekursion är iteration (loop)  
  Istället för for,while eler do så används upprepade anrop till den egna metoden. Varje anrop förminskar problemet. Tillslut finns bara ett enkelt program kvar. En enkelt svar som kan lösas utan rekursion villket också gör att rekursionen når sitt stoppvillkor.
* Parametrar och metodvariabler är lokala. Dessa finns så så länge metoden är aktiv
* Vissa problem lämpas sig för rekursivlösning. Men andra har en iterativ reduktion. Rekursionen tar plats på anropstacken. En rekursiv algorith kan vara svår att förstå. Även om den är elegant. (Prydligt)

*F12 – Collections*

* En Collection är ett objekt som fungerar som en samling av andra objekt  
  Lägg till objekt(add, put)  
  Ta bort objekt ( remove,clear)  
  hitta objekt (Contains, indexOf, get)
* En collection är en samling data och operationer på dessa data. Tillsammans utgör de en abstract datatyp.
* Ofta är alla element i en collection av samma typ.
* Kompilatorn kan hjälpa med detta. Då måste man ange vilken typ man tänker sig på elementen.
* ArrayList  
  ArrayList är en samling baserad på en array. Objekt som adderas placeras i arrayen. Om arrayen tar slut -> skapas en ny längre array. Arrayen behöver alltså inte växa så ofta.
* Kö – Elementen läggs till sist och tas ut först.
* FIFO – First-In-First-Out
* Stack – Elementen läggs till först och tas ut först
* LIFO – Last-In-First-Out